# EPISODE 1520

[INTRODUCTION]

**[00:00:01] ANNOUNCER:** This episode is hosted by Lee Atchison. Lee Atchison is a software architect, author and thought leader on cloud computing and application modernization. His most recent book, *Architecting for Scale*, is an essential resource for technical teams looking to maintain high-availability and manage risk in their cloud environments. Lee is the host of his podcast, Modern Digital Business, and engaging and informative podcast produced for people looking to build and grow their digital business with the help of modern applications and processes developed for today's fast-moving business environment. Subscribe at mdb.fm. And follow Lee at leeatchison.com.

[INTERVIEW]

**[00:00:45] LA:** Observability is a critical aspect of modern digital applications. You can't operate an application at scale that satisfies your customer needs without understanding how the application is currently performing. Whether it's understanding the current operating needs of the application, adjusting resource usage, detecting issues before they become serious, or solving an ongoing technical issue as it's going on. Application observability is a critical aspect for all modern applications.

Thundra is a performance monitoring company that specializes in application performance monitoring, infrastructure monitoring, and most notably, serverless monitoring. Berkay Mollamustafaoglu is the CEO of thunder. And he's my guest today. Berkay, welcome to Software Engineering Daily. And I have to start out by apologizing for completely mangling your last name. Why don't you tell everyone how it should be pronounced?

**[00:01:43] BM:** Sure. It's Mollamustafaoglu. A handful combination of three words, but I appreciate you tried.

**[00:01:51] LA:** Mollamustafaoglu.

**[00:01:52] BM:** Yes, that's right.

**[00:01:53] LA:** Not bad. Okay. Okay. Well, we'll stick with that. I won't try it again just in case I get it wrong even worse than I did the first time. I apologize for that.

APM. So, there's lots of companies today in the application performance monitoring space. There's the Datadogs and Dynatraces, which are the biggies according to Gartner's magic quadrant. But there's also the New Relics, the Splunk, the AppDynamics and other companies like that. What makes your APM offering so unique?

**[00:02:23] BM:** Yeah, that's a very good place to start. APM is, in a sense, a generic way to approach this monitoring challenge that we need to monitor, as you said, the applications to know what they are going on.

At Thundra, the origin is the serverless, as you also mentioned. And that came out, the Thundra team. We were in OpsGenie, in my previous startup, where reliability was a very, very high importance for us. And therefore, observability monitoring was important. And we were moving some of our infrastructure to the serverless environment very early. And we couldn't find any tools to get visibility into what's going on in that environment to the code that we were deploying there.

The team proposed that we developed something. And we were internally just developing for our own needs. And OpsGenie was acquired in 2018 by Atlassian. And prior to that acquisition, we agreed to spin off the product as a separate company, Thundra. That's how it came about. And the team moved on to Thundra to further build basically the product for the market.

It was from the beginning monitoring solution for the specific use case. And in this case, the use case was serverless. Serverless is a very different environment. You don't have servers to put agents on in a way. It's very lightweight. You're talking about milliseconds of execution time. So, it needs to be very rapid. And also, at the time at least, a lot of the solutions had split different aspects of monitoring. Traces in one place. You had the tracing solution. There's a logging solution where you got the logs and the metric kind of a solution. There's, often, very different products. And it was very hard for the engineers to switch between these products depending on what they needed.

We started with those two principles. One, it needs to be all automated. Automated instrumentation of the code, in the application level, as opposed to tagging manually, try to follow things, which was often the case in distributed tracing. It needed to be lightweight. Anything to work in the cloud, AWS cloud and so on, where serverless was most common and what we were using at the time.

We started from those principles. And the other one was that it needed to have all aspects of observability in one place, traces, logs, matrix, that you could see everything in context of each other without switching between tools. Those were the guidelines we had sort of started with, and build it – At the time, when we built it, there was almost no product that did all these things in one place.

**[00:05:18] LA:** That actually is a unique way to look at it where you don't want to require custom instrumentation within the application. I think that is one of the things that definitely makes you unique from some of the other major vendors. But is that easier in the serverless world than it is in the monolith world? Because, essentially, it's impossible in the monolith wealth world to go completely **[inaudible 00:05:41]**, right? Without having some sort of instrumentation. But is that easier in a serverless world? Or are there other challenges there?

**[00:05:49] BM:** I mean, there were some things that made it easier. But there are definitely challenges. It needs to be really lightweight. If you put a lot of overhead to it, your costs would double because you're taking more time. And like it's you're paying for the compute time and how much memory you're using. It needed to be like existing agents putting into the server that's just didn't work. Some customers were trying that. So, that making it very lightweight made it harder.

We focused on the AWS Cloud initially. And they made it a little easier because it was well-defined kind of services that we could look into. But without tagging doing distributed tracing was very difficult. And I think there may be a couple products in the market still that can do that without manual tagging of the stuff. Like, when you put something to a queue and take it back from queue, or put Kinesis and take it out, they typically lose the transaction.

Thundra was, from the beginning, able to stitch those testing automatically and give you a full view of the transaction. I think that's probably what we spend the most time getting it right. But because it was for a specific environment, the cloud services and so on, we were able to do that as we focused on it. In time, I think the infrastructure providers like AWS Cloud providers will provide more capabilities to help the vendors to do this. But in the past up, until now, it was kind of left as an exercise to the vendor. And I think we did a really good job at that.

**[00:07:26] LA:** Yeah, I know that was always a challenge. Full disclosure to everyone, I came from New Relic. I was there for a number of years. But when we tried to do this in New Relic, we absolutely took the approach of instrumenting the code. Ran into all the issues you were talking about. And one of the things I always thought that we should have done more of, and it sounds like maybe this is the approach that you took, is to leverage the data that AWS is generating for you. And they've obviously done more of that as time has gone on. But that was something I always thought was an important aspect in a complete monitoring solution is to integrate in the Cloudwatch characteristics that were built into native into AWS in a much fuller manner. Is that how you're able to do it? Or is there more to it? There's obviously more to it than that.

**[00:08:18] BM:** There was more to it, yeah. I think going forward, I believe there will be more in Cloudwatch that will help with this to all vendors and the customers doing this. But Serkan is our CTO and the brains behind this. He's kind of a people that knows the internals of all these things quite well in the cloud as well as inside the application runtimes. And by combining, stitching in the backend with the data that we collected, like the techniques there, as well as smarter agents, fairly sophisticated ways, we were able to do it. It certainly wasn't easy. And like I said, it's one of the more innovative guys there. And having very deep knowledge of this in terms of the systems, he was able to do it. I can't say that I fully understand everything he did to make it work. But he did a great job at getting it done.

**[00:09:11] LA:** Got it. Got it. One of your core values is as much stitching together of the transactions that is possible without deep integration into – custom integration into the application.

**[00:09:27] BM:** Yeah, one of one of the things of the principles of serverless is to take load off of the developers, right? Like, you don't manage the infrastructure and so on. And if you just turn around when you're trying to do observability, and now you have to instrument your code manually, it just is not in sync with the philosophy of the serverless, in my opinion. We are trying to take load off of the developer's plate. We're asking a lot from the developers now. As you know, they have to think about everything now. How to run it? How to secure it? And so on.

We really thought we can't ask anything manually. When you deploy this, it should work. We weren't sure that we could do in that level. But the team kept at it. And eventually, we found enough information how to stitch the codes. We use Elasticsearch in the backend and being able to like – essentially, the data came as separate parts. But we were able to find sufficient data to be able to connect the different traces that are related at the backend with some processing.

And agents provided some of that data that we collected from the Lambda functions executions and so on. The data is there. But it's just very obfuscated. We were able to eventually stitch it together. It's fully automated. But that was the driving sort of a principle. People are trying to reduce the overhead observability and like computing and so on by going to serverless. You can't turn around and ask them to do more work, to do monitoring of that environment.

And I think that is the right approach, that there will be more and more as time goes, by provided it will make it easier by the cloud providers to do those things. And we're trying to close the gap in the meantime to provide a solution.

**[00:11:20] LA:** You've talked mostly about AWS so far. And I know that's where your start was. What about Google? What about Azure? Do you have offerings there? Or do you have plans for offerings there? And we can then have a conversation about hybrid cloud as well. But, your thoughts there.

**[00:11:35] BM:** Yeah. Our solution initially, as I mentioned, was serverless. But later, we found that a lot of our customers were running hybrid. It wasn't pure serverless. They were running containers, VMS in some cases. We did extend the solution, the APM solution, to support the containers and VMS as well even though it's still the majority of the customers that come to use Thundra. They come because they have a significant serverless. But it's not just serverless they have. But because we approached this from an application level. Instrumenting the runtimes. It does work in Google and so on as well. But the number of customers we have in those environments is very, very small percentage, single digits. It's just – because how we are known, or where people are doing serverless more and so on. Vast majority of the customers are in the AWS environment.

**[00:12:35] LA:** What we're finding is there's a lot more customers too that are turning to, well, polycloud applications. These are you know applications that are in multiple cloud providers not as a repetition but using unique capabilities of each. You might have your AI capabilities in one cloud provider, your data storage in another, etc., etc.

So that naturally means you have interactions that go back and forth between the cloud providers. How do you deal, or can you deal with, or this a hard issue to solve with dealing with tracing across cloud providers like that?

**[00:13:11] BM:** It will be an interesting challenge. We haven't done much of that. Just simply, we didn't have customers demanding that yet. They do have applications in multiple clouds, but not a lot yet in between them. And when there are, as long as it's in the web HTTP level and so on, we do trace. Like, it doesn't matter which cloud they are on. You're able to see the trace regardless of where they are.

Well, we haven't had anything that required us to do specific things for this. Maybe the customers are doing it, I just haven't noticed that. Because there is no inherent limitation preventing that. I just don't have first-time knowledge of it.

**[00:13:50] LA:** Got it. And it's not some something you've had to specifically design for. It just works with what you're currently doing.

**[00:13:58] BM:** Yeah. Yeah, it works in hybrid environments. There is no limitation that it wouldn't work. But again, the vast majority of the customers we have typically come to us because the generic solutions, they're lacking in some ways, right? There are other solutions that do those things. And if they are coming to a startup that is not as well-known as, say, New Relic and Datadog, they typically are looking for something more. And I think in the AWS serverless world, we have that much more. In the other spaces, maybe we are similar in it. So, that they're not gravitating to us. But if you are doing serverless and you want full visibility, observability for that environment, it is, I think, the best in the market. The customers that who have that need, they run critical workloads and so on, those are the ones they typically come to us. Because it's just not enough to do basic monitoring of serverless environment with other tools that you might get done.

**[00:15:01] LA:** What about a hybrid? Customers with cloud and on-premise components of their application, do you see that much? Or again, is it mostly people who come to you? They're mostly focused on the cloud aspects of the monitoring?

**[00:15:16] BM:** I mean, there are some, but mostly cloud is one. We haven't seen a lot of – Our customers are, I would say, startups and mid-market. Not the enterprise customers. They seem to have more, more of that like on-prem/cloud. Our customers are – Even in the, say, more traditional companies, they are the teams that are adapting more sort of a cutting-edge stuff. They are doing new Greenfield applications in the cloud using serverless and adapting new tool sets for it and come to use our products. Versus replacing may be the traditional tools, APM tools, that they have for their hybrid stuff. That may come later. But so far, it's been more the new stuff that they want to have solution for is why they acquired our products.

**[00:16:07] LA:** Got it. Got it. That makes sense. There's a big focus nowadays on cloud native applications, which is much more than just built for the cloud. It's microservice applications. It's containers. It's Kubernetes, etc. Serverless is kind of orthogonal to that, but very much related to that.

But I do know there is a – I don't know if the word debate is the right answer or not. But there's a viewpoint of the value of containerized microservices versus serverless, which from an architectural standpoint, the pros and cons of each and how they all work together. Applications, really, truly, most applications, cloud native applications, end up with some combination of both of them. And so, monitoring across containerized applications – containerized services, I should say, and serverless services, becomes critical. And so, tracing works across that environment. And you may call that work correctly.

**[00:17:08] BM:** That's right.

**[00:17:08] LA:** Yeah. Oh, that's great. What do you find your customers focusing on more and more? Do you see them – I try and teach my customers that serverless is great for certain types of problem sets. And everything else container-based microservices is the preferred architecture. And I do know some customers though who say, "Nope, serverless is everything. Put everything in the serverless." And others who say, "No, no, no. You can't use serverless at all. Go go away from that."

Like I say, what I focus on is use serverless for the things it's good for, and that's it. What do you find from your customers? Are there people who are all-in on serverless and that's it? Or are they truly trying to find the balance between the two?

**[00:17:54] BM:** If they're starting new, we have a lot of customers that went all-in on serverless. They use some containers. But that's the exception for them. If there is a reason for whatever reason that it's better for it, then they got that.

The traditional companies, the more established companies, let's say, typically do the other way around. And they have more containerized applications and use serverless more of an exception.

I think the primary reason has been that. The serverless had been lacking some technologies that makes it easy to lift and shift existing applications. The containers made it very easy. It's not a huge leap to go from running in a physical server or a VM on-prem to like moving to a container. It's been a significant change if you wanted to move that to the serverless environment. Sometimes the savings are worth it. Often, it's not.

That's why like I think we've seen a lot more move by the customers to the containerized environment to that approach. But if you're building something new, operational advantages of serverless is significant. But it still requires more of a change in how people develop applications. How they run operations? And so on. It's not container-based stuff. It's more incremental change, I would say, to the way that people do things. Serverless demands more of a change.

And tooling has not been ready in all those areas to accommodate that. In a way, we are trying to fill some of those gaps. In addition to our APM product, we experimented a lot about the debugging in serverless environments. We have built a serverless debugger product. Initially, it was part of the APM suite that we now split it as a separate product that you can get for debugging serverless applications in production. Like, put breakpoints like you would do in a regular application and inspect and so on. Being able to debug those things. That's another Thundra product.

We are also working on development product where you can develop the Lambda in your laptop, or in your machine, in your IDE, but run the rest of the environment on the cloud. So, you don't have to simulate everything in your laptop. Or you don't have to deploy every time you make a change and so on and deploy to the server, which is another, I think, barrier in adoption of the serverless. Because you couldn't just use your IDE, debug on your ID and do all the things that you do. And you have to move to a different environment, which is not – I think it's a little asking too much. Our development solution that we open source them. Building it to make it available as SaaS as well. We'll address that so that it's just making it a little bit easier.

In time, again, I think the serverless will provide more of the functionality the customers need to lift and shift applications to serverless without changing the entire way they work. But I don't think it's there yet. And as a result, I think a lot of whom are choosing the containerized environments to run their workloads.

**[00:21:27] LA:** Yeah, yeah. And that's certainly something I hear a lot too, is tooling for serverless is getting better. It's getting much better than what it was in the early days. But it's still got a long ways to go.

One of the other things I specifically hear a lot about, and it sounds like you're in a good position to deal with, is one of the downsides of serverless compared to a more traditional cloud native application is performance predictability of serverless, is the delta in performance of a given function varies considerably more than it does in a containerized environment. I'm wondering, first of all, have you seen that? And are you able to help customers deal with those issues?

**[00:22:08] BM:** Yeah. I mean, the area that we've seen that the most is what people report as the cold start, especially for applications that don't have a lot of load or very varying laws. And when you have to wait for a container to warm up, like you get these massive delays.

In our own applications as well, we've run into this. And especially in like Java, for example, it was significant. We've done a lot to solve that inside Thundra and outside, like, to keep functions warm and manage all that. It got better in time. And I know that the AWS is working on some significant improvements on that that is coming, I think, in the next few months, which will again help.

But so far, when we were building some API-based stuff, the other side expected an answer in three seconds. And we couldn't guarantee that to Lambda. We had to put a container there, because it was just sometimes taking more than that. You couldn't take your Java application and put it into container and stick it to serverless and expect the performance. Cold start can be 10 seconds. It wasn't unheard. That's where we saw the most of the problems.

It got better in time. It's less of a problem in some run times than the others. But still, I think something that will need to be in much better shape for serverless to be an acceptable solution for more type of applications.

**[00:23:54] LA:** I completely agree with that. And we find that TP99 performance for a lot of applications to be very, very poor with serverless because of this cold start syndrome. Every once in a while, you lose some containers. They'll go away. You need to increase the number of background containers available. And it's not just a matter of keeping individual serverless functions always performing. In other words, if you just leave it set for an hour, when you come back, the first call is going to be very high. It's not that simple. W

What often happens is there might be 20 instances of this function that are warmed up, but you now, for some short period of time, need 22. And so, two new instances have to start up. There you have to warm, go through their cold startup process. And so, randomly in the middle of an application you get these very, very big spikes.

And those are tough problems to deal with. But one of the other things that I've seen customers have to deal with isn't the TP99 cases much as the larger standard deviation. On a containerized application, you can build a – of course, it all depends on the application and the **[inaudible 00:25:09]** space and all that stuff. But you can build very, very, very real-time functions that have very predictable performance with a very, very tiny standard deviation. But even those sorts of functions, we move them to a serverless environment. Forgetting the warm cold start issue, the standard deviation can still be a lot greater. Is that getting better too as time goes on? Are AWS dealing with those issues more and more?

**[00:25:35] BM:** It is. But I think there is a case of you don't know what's going on. And those are the parts of the system that we don't have a lot of visibility of why that happens. And also, in case of cold start, we don't control how many warm things are. We wanted those things to be able to explicitly control, like, let's create something and so on. But beyond that as well, since it's automate, I think, in the container-based stuff, that you have more control over where the time is spending and it become more deterministic.

A lot of things happen magically in serverless world. But when it doesn't happen, or doesn't happen the way that you expect, you have no recourse. Things that are sensitive to latency or how long it takes, just like you, I'm kind of suggesting to stay away as using the serverless in those cases. It's just there needs to be more control and visibility to understand why those things happen and eliminate those deviations. They are there. But they are definitely better, significantly better, than the early times. When we look at the performance metrics, we can see that. Whatever they're doing, it's working. It's just they need to do more of that.

**[00:26:56] LA:** Yeah. In the very, very early days of serverless, I remember looking at some performance charts. And it looked like white noise as far as their ability. It's like no predictability at all. But it is getting a lot better now. That's certainly true.

Let's go on with the idea of complexity a little bit more. Certainly, the idea that serverless applications can be more complex than cloud native or than certainly the monolith from the standpoint of the infrastructure. Forgetting about the complexity of the code itself, but the infrastructure. And one of the places where you see that complexly show through is in the setup integration deployment, pipeline process of getting the application running and upgrading and things like that.

Now, not very many performance monitoring companies pay attention to the pipeline. But you have. So, can you talk a little bit about what you do from the performance pipeline standpoint? Or the deployment pipeline standpoint?

**[00:27:53] BM:** Yeah, yeah, CI pipeline. Actually, when we talked, who is Thundra? We are really now observability. But like our expertise is in automated instrumentation and monitoring of complex solutions. Serverless was one specific example. Like, it was a different environment. It had different needs and constraints. So, we did that. But another area is the CI pipeline.

The process of moving code from GIT to production, it used to be like you ran a command, built it, and it deployed it. It was fairly simple. Now, you look at it, and it's a monstrous application with lots of different moving parts. Some of the shelf applications, some of the source solutions, SaaS products, and the pipeline kind of orchestrates it. It's a very sophisticated, complex environment. And we realized that like there was very little visibility into what happens in that environment. When things are slow, your build takes long. Your tests are taking long. You didn't know exactly why.

And applying just generic APM solutions, just like it didn't work for serverless, it didn't work in this environment either, because it doesn't understand what the test is. And what is the coverage reporters? And so on

We thought like it is a similar problem that we can add some value. So, we looked at. Wwe applied our know-how in technology in terms of automated instrumentation, tracing and understanding of different type of data, first, to provide visibility. You can tell like how long it took to run the pipeline. Where do you spend the time? If it failed, why it failed? Typical, it fails. You run it again. It maybe passes this time. That type of thing.

Or on the – We talked to a ton of customers, and the only way they were debugging, troubleshooting, was logs. So, kind of, before APM where things were in 20 years ago? 15 years ago? You'd used a lot of log statements and so on. That's the only way you understood the behaviors.

That's where we found the CI pipelines. The only courses like if you're logging stuff, you kind of dig around and figure out where time is spent. We decided that like this is definitely not where things should be. And there wasn't anything in the market doing this. We decided that like if we built something, that it understands the CI pipelines. Understands what a test is. What you set up before the test? And how long the test runs to set up? And then to clean up afterwards and so on. And you trace the tests when things are okay. And then when it fails, you can compare the failed test flow with the run. You can much faster go to the root cause of why the test fails.

And also, when PR comes in, you know what changed. The PR tells you, "Well, these are the lines that changed. These are the new lines." We can kind of look, "Okay. Well, these are the things that changed. These new changed things, are they covered by the tests? Or is there stuff that is not covered in this PR?" Based on that, you can have different flows in your software reliability and so on.

We looked from both resource utilization perspectives and the failures and performance trends of which tests are taking long. Where do you spend the time to like deeper? Are you doing the things that you expect to do? Is your coverage there? And so on. And we'll go deeper into getting deeper understanding of what happens in CI pipelines in time, so that you have a reliable pipeline chain to move the software to production. And you can know when things are not right. Like, why they're not right? And quickly address that.

**[00:32:03] LA:** It sounds like there's a lot of fertile opportunities in that space in particular. There's certainly not a lot of people dealing with those issues right now.

**[00:32:11] BM:** True. There are only a couple vendors even tackling that. Datadog is one, I think. We are doing it. There's some working on it. But there isn't really – It's a new area. And I've been in monitoring space for a long time. It was one of my first jobs. And it just stuck around. I worked in the customer side and eventually moved to the vendors to do this. And it reminded me, really, like, application monitoring was like this. We didn't know.

We would look the server utilizations and the logs in the servers and stuff, but we didn't really know what was happening inside the applications before the lose started in CA with the wileys and so on. Try to look into the applications. BMC patrols of the time and the CA and so on. And then much better with the New Relic and so on.

But when I looked at the CI list, it was like the dark ages of application monitoring. You didn't really know what was going on inside. I found perilous to it, like, "Yes, you can." Really, it's very opaque. You don't even know what's happening inside. You look from outside, like it took this much. We look at builds that are taking more than an hour. And does it need to take more than an hour? Or is it just poorly organized? It's hard to say. They don't have that information.

When they say, "Okay, we want to reduce this." They have to go then manually put stuff to measure things and figure out like what can be parallelized? And so on. And this build time, it turns out, has a very direct impact on how successful an organization is. The longer the build times and the more frequent the builds fail, the less the deployment frequencies, which is one of the four-dollar metrics of like how in the maturity in the DevOps. You really need to be able to deploy faster. And if you have an hour build times, it's very difficult to do it. Especially if the failure rate in those bills is high. Like, the people are waiting. Our guys call them master wars. They're waiting for the others to like push so that they can push their code. And a lot of waiting around and fighting going on just to get to production.

When I saw that like how much time is being wasted, the valuable developer time with this, optimizing that pipeline, making sure that it takes short. And when there's a problem, whether it's performance or a failure, you can't understand why it is. And you don't just run the test again and hope this time it passes. The flaky test, so to speak. Those are not – they have no place in the modern, I think, software development. We are coming from behind as a sector. We thought, like, given our expertise in like instrumenting and monitoring complex environments, this is an area we can add value. So, that was kind of the second product that the team came up with we call Foresight.

**[00:35:28] LA:** Serverless environments, monitoring, serverless debugging, cloud APM, and CI/CD performance monitoring. It's quite a wide variety of offerings that that cover some very important and critical areas. But what's next for you?

**[00:35:47] BM:** We have one more product that we launched a month ago as an open source project called Sidekick. I am very excited about this one. Particularly, I think because it's going sort of against the current tied in the monitoring observability world.

Because our technology, I feel like, supports now collecting large amounts of data, it's cheaper technology. We can do tons of this. We chose now as an industry collect everything, instrument everything, collect all these data. If there's terabytes, if not more of instrumented data, logs, traces, metrics. And then you can use big data methods to make sense of it, right? That's kind of where we moved.

And maybe because I work with these times where things were more constrained. Like, you couldn't collect all the data. You had to do intelligent sampling. You have to do all kinds of things. You couldn't put a lot of overhead into the servers that application is running and so on. I found this approach, like, I get to appeal. Like, when you have everything and you go back, you can look at all the data and figure out what's going on.

But I feel it's not feasible for all applications or workloads. It's for maybe the most critical applications, you can do that. But for a lot of applications we were finding in serverless world as well, the cost of monitoring and observability can be as much, if not more, of the how much it costs to run the environment. It's just very hard to explain to the business like why you're paying thousand dollars to run the application and thousand dollars to monitor it. It's just because of the data that generates it so much more than the application data that the application itself generates.

I was always like a little skeptical to this. Like, yeah, put everything into these big data sources and then use tools to harvest them, or like mine them? And so on. The approach that we've sort of adapted from what we call live application debugging, essentially, when you're building developing an application, how do you find problems? Like, you put breakpoints into the code. You look at the stack and so on. Very deep data on what the application state is.

What if, basically, you could do this in production, but only when there is some signal suggesting that there may be something wrong instead of doing it all the time? Maybe not a great analogy. But we don't do those COVID tests that you stick into your nose all the way to your brain like every hour just in case, right? We do those tests when you have a symptom. Then you do that, and maybe the blood test, and like much more data. Because you have some kind of a signal that you may be sick. That's kind of how I look at the approach that we took with Sidekick. You can monitor the application help. And if there is anything, the response time is slow, or there's an exception somewhere in the application and so on, then you can trigger very deep data collection. And then compare the data you collected with the time that when everything was normal. This is kind of how we troubleshoot things. What changed? And how is it different. This is what changed. Recent changes. Foresight through the integrations tells us this is what changed. These are the PRs. What parts of the code changed? What tests are impacted?

But any system, like a synthetic transaction stuff, real user monitoring, whatever, that as they are monitoring from a customer's perspective, user's perspective, if there's something detected that, like something seems off, then they can trigger deep data collection. And we compare that to after a deployment, when everything was working, we collected the data. Well, this was the normal state. This is the abnormal state. You compare to find where the problem might be originating from.

**[00:40:07] LA:** So, you enable a deeper level of testing. Probably you do it right after you deploy, but also whenever you notice a problem, then you have something that you can compare it to. Can you do that automatically?

**[00:40:19] BM:** Yeah, there's some you can do automatically. Some requires some help from the developers. Basically, developers can say, like, "If you notice something like this, make sure you collect data in these disease points in the code." They can put those breakpoints, essentially what we call trace points, into the code. And when there's a signal, we collect that data.

But we also, in truth, try to guess like where we should collect the data without the manual work as well. Then I think we will get better at that. If this problem happens, I want to collect data in this file and that file. Because those are the files that changed after the last deployment. If something is a miss, it's more likely to be there. Let me collect some data type stuff.

We started rule-based. And it's, again, magic machine learning or AI. Yeah, at least not yet. So, you can basically say, "If this kind of failure signal, then collect this type of data." We are building the APIs now so that it can be triggered by any system, like any monitoring solution, or testing solution, etc., can say, "Collect this data and send it to me."

If you're using Splunk for your operational system and logging, and a monitor detected a slowdown in API, you can trigger this data collection and we send it to Splunk. And an advantage is that it can be directly linked to the alert. You don't have to mine gigabytes of data. This was the base. This is what is now. Compare and figure out what it is.

**[00:42:00] LA:** Right. It's a great idea. I think the risk, of course, is that you create a complexity in the triggers and the the conditions. And when do you do for what the circumstance? How you do in other circumstances? And so, it's a prime candidate in the future for an AI solution to help with some of that. But this is still a world's better than what we have right now, where like you say, you either collect everything or you collect nothing. You can't manually collect the right things ahead of time unless you collect everything.

**[00:42:33] BM:** That's why we also open source this. Because I think as a method, an approach, or philosophy, I think it needs more thought and help and so on. We needed in our arsenal. I don't sort of subscribe to like this is the best way. Like throw everything else. And like do it this way. But I don't think the current approach is suitable for all applications.

And now, with the economies going where it is, and the budget's becoming more under pressure, I think I'm kind of concerned that we'll see a lot of pullbacks because the ROI on investment and observability monitoring will be harder to justify because it's very expensive. Where in the boom times, we didn't care as much. Just grow and make sure that things are reliable. Now I see a lot of people are like, "Why am I paying this much? And what is it getting to me?"

I think as the industry in our arsenal, we need this as an approach, where you have simple monitoring of maybe user 's perspective. Like, the performance, errors and stuff. And then that triggers when there's a symptom. Just like you seem to be sick, then we do all the tests to figure out what's wrong. That approach, versus we're doing tests all the time and collecting all the data for everybody. That's not sustainable, I think, for all workloads. A lot of workloads, it's overkill and it's too expensive. Hard to justify the cost. That's why we decided to open source and then see where it goes with the different things that people might come up with it. That's the next thing that we're working on.

**[00:44:19] LA:** It's going to great to see what happens with this. Because I think it's a great area and a great – Like you say, I think we need a lot of research in that area, and to be it's a great fertile area for some open source contributions. I wish you luck with that. That's great.

**[00:44:33] BM:** Thanks. Thanks. Exciting stuff.

**[00:44:35] LA:** Yeah, definitely, definitely. My guest today is Berkay Mollamustafaoglu, who is the CEO of Thundra. Thundra is much easier to say than your name. But thank you very much. And thank you for being on Software Engineering Daily.

**[00:44:49] BM:** Thanks, Lee. I appreciate it. It was fun.

[END]